GradeMax



Mark Scheme (Results)

Summer 2023

Pearson Edexcel International GCSE
In Computer Science (4CP0/2A)
Paper 02 Application of Computational Thinking

Edexcel and BTEC Qualifications

Edexcel and BTEC qualifications are awarded by Pearson, the UK's largest awarding body. We provide a wide range of qualifications including academic, vocational, occupational and specific programmes for employers. For further information visit our qualifications websites at www.edexcel.com or www.btec.co.uk. Alternatively, you can get in touch with us using the details on our contact us page at www.edexcel.com/contactus.

Pearson: helping people progress, everywhere

Pearson aspires to be the world's leading learning company. Our aim is to help everyone progress in their lives through education. We believe in every kind of learning, for all kinds of people, wherever they are in the world. We've been involved in education for over 150 years, and by working across 70 countries, in 100 languages, we have built an international reputation for our commitment to high standards and raising achievement through innovation in education. Find out more about how we can help you and your students at: www.pearson.com/uk

Summer 2023
Question Paper Log Number P72938A
Publications Code 4CP0_02_2306_MS
All the material in this publication is copyright
© Pearson Education Ltd 2023

General Marking Guidance

- All candidates must receive the same treatment. Examiners must mark the first candidate in exactly the same way as they mark the last.
- Mark schemes should be applied positively. Candidates must be rewarded for what they have shown they can do rather than penalised for omissions.
- Examiners should mark according to the mark scheme not according to their perception of where the grade boundaries may lie.
- There is no ceiling on achievement. All marks on the mark scheme should be used appropriately.
- All the marks on the mark scheme are designed to be awarded. Examiners should always award full marks if deserved, i.e. if the answer matches the mark scheme. Examiners should also be prepared to award zero marks if the candidate's response is not worthy of credit according to the mark scheme.
- Where some judgement is required, mark schemes will provide the principles by which marks will be awarded and exemplification may be limited.
- When examiners are in doubt regarding the application of the mark scheme to a candidate's response, the team leader must be consulted.
- Crossed out work should be marked UNLESS the candidate has replaced it with an alternative response.

<table>
<tr><th>Question</th><th>mp</th><th>Answer</th><th>Additional Guidance</th><th>Mark</th></tr>
<tr><td>1(a)</td><td>A1
A2
A3</td><td>Award one mark for each correct cell:

<table><tr><td>Input</td><td>Debit / Credit / Description</td></tr><tr><td>Processing</td><td>Balance + Credit / Balance - Debit</td></tr><tr><td>Output</td><td>Balance</td></tr></table></td><td>Input: accept examples of inputs e.g. 10.23 (must be in Figure 1)

Processing; mark first 3 things (ignore extra text) if correct e.g.
balance + credit – debit (1)
balance + credit + debit (1)
balance – credit (0)
accept examples from table e.g. 128.35 – 10.23 (1)

Output: accept description of formula e.g. subtract debit from balance (1)</td><td>3</td></tr>
</table>

Question	mp	Answer	Additional Guidance	Mark
		Award **one** mark for each of:		
1(b)(i)	B1	This is 5%		**1**
1(b)(ii)	B2	or Accept || for C# and Java		**1**
1(b)(iii)	B3	grossAmt netAmt		**1**
1(b)(iv)	B4	TAX		**1**
1(b)(v)	B5	inGross		**1**

Question	mp	Answer	Additional Guidance	Mark
1(c)(i)	C1	Award **one** mark for: To store a value that will not change (during program execution) (1)		**1**
1(c)(ii)	C2	Award **one** mark for any of the following: • It is less likely to be changed by accident (translation will error if attempt assignment)(1) • The value only needs to be changed in one place (for maintenance) (1) • It is easier to read/understand **the logic** of (1) • Ensure consistency of value if it has a name rather than the hard-coded same value repeatedly (typo) (1) • Can make code more efficient (optimisation / value stored rather than reference)	MP1 And MP4 are referring to programmer. Do **not** accept answers relating to user.	**1**

Question	mp	Answer	Additional Guidance	Mark
2(a)(i)	A1 A2	Award **one** mark for any of the following up to a maximum of **two** marks: • The code in the library has already been debugged/tested/optimised / library code less likely to have errors (1) • May include specialised/proprietary functions (1) • Library subprograms can be imported into code when needed / don't need to write code themselves (to save a programmer time/work) (1) • making it more readable / makes code easier to understand (Code files become shorter)	Allow characteristics of a specific library (math, random), if it can be attributed to unique bullets Don't allow quicker/easier/better without a reason being given Accept answers about subprograms as candidate may have written these themselves Line numbers are for guidance – mark answer as a whole	**2**
2(a)(ii)	A3	**The only correct answer is A** ***B*** *is not correct because iteration is not callable and does not return a result.* ***C*** *is not correct because a procedure, while callable, does not return a result.* ***D*** *is not correct because selection is not callable and does not return a result.*		**1**
2(a)(iii)	A4	Award **one** mark for: • A variable that exists/is accessible only in the subprogram/scope in which it is created (1)		**1**

<table>
<tr><th>Question</th><th>mp</th><th>Answer</th><th>Additional Guidance</th><th>Mark</th></tr>
<tr><td>2(b)(i)</td><td>B1
B2
B3</td><td>Award one mark for each correct row.

<table><tr><th></th><th>Adult</th><th>Children</th></tr><tr><td>Normal</td><td>1/2/3/4</td><td>1/2/3/4/5/6</td></tr><tr><td>Boundary</td><td>1/4</td><td>1/6</td></tr><tr><td>Erroneous</td><td>Any value <1 or >4</td><td>Any value <1 or >6</td></tr></table></td><td>Accept all valid values for normal and boundary
Do not accept written numbers (two) as question asks for ‘numeric data’ – this is testing range, not data types.</td><td>3</td></tr>
<tr><td rowspan="5">2(b)(ii)</td><td colspan="2">Award one mark for each of:</td><td rowspan="5">Runtime error (B7) could be fixed before logic error (B6)
Applies to all languages
Accept < for <=
Allow /-1000 as alternative to B6 /1000 could happen after initial location e.g. print(“Order”, weightNeed/1000, “kilograms)</td><td rowspan="5">4</td></tr>
<tr><td>B4</td><td>Change =+= to +</td></tr>
<tr><td>B5</td><td>Change > to <=</td></tr>
<tr><td>B6</td><td>Change order of subtraction to: requiredWeight - inStock</td></tr>
<tr><td>B7</td><td>Change 0 to 1000</td></tr>
</table>

Question	mp	Answer	Additional Guidance	Mark
3(a)	A1 A2	Award **one** mark for any of the following up to a maximum of **two** marks: • Each column represents a variable/output (1) • Rows shows how the values of variables change (as the code is executed) (1) • Values are filled in manually/by hand (1)	Line numbers are for guidance – mark answer as a whole	**2**

Question	mp	Answer	Additional Guidance	Mark
3(b)	B1 B2 B3	Award **one** mark for each correct cell: **num1** | **num2** | **Output** 88 | 18 | State 2 17 | 18 | State 4 12 | 19 | State 1	ignore quotes e.g. “State 2”	**3**

Question	mp	Answer	Additional guidance	Mark
3(c)	Award **one** mark for each of:		C4 should exit loop Allow C5 if a loop is used to keep asking for a non-negative exponent C9 – ignore error message in C6; must have base, exponent, and answer in output for this mark	
	C1	At least one variable with a suitable variable name		
	C2	At least one variable initialised to an appropriate value		
	C3	While loop used to continue running code		
	C4	Relational test to identify terminating condition for loop		
	C5	Selection used to check for negative exponent		
	C6	Appropriate error message if exponent is negative		
	C7	Calculation of answer using any method		
	C8	A loop used to calculate answer as indicated in flowchart		
	C9	Prompts and output messages are fit for purpose		
	C10	Executing and producing the correct output for anticipated inputs		**10**

Suggested test data and output		
Base	**Exponent**	**Output**
0		ends program
2	0	2 to power of 0 is 1
3	4	3 to power of 4 is 81

C#

```
// Initialise variables
int baseNum = 0;
int exponent = 0;
int answer = 1;
String outString = "";

// Display the first prompt and get the base number
Console.Write("Enter a base number (0 to exit)): ");
baseNum = Convert.ToInt32(Console.ReadLine());

// Check if user wants to exit
while (baseNum != 0.0)
{
    // Display a prompt and get the exponent number
    Console.Write("Enter an exponent number: ");
    exponent = Convert.ToInt32(Console.ReadLine());

    // Check if the exponent is negative
    if (exponent < 0)
    {
        Console.Write("Invalid exponent entered");
    }
    else
    {
        answer = 1;

        // Calculate the answer using exponentiation
        for (int i = 0; i < exponent; i++)
        {
            answer = answer * baseNum;
        }

        // Create a string for the output and show the user
        outString = baseNum.ToString() + " to the power of " +
                    exponent.ToString() + " is " +
                    answer.ToString();
        Console.Write(outString);
    }
    // Display the first prompt again and get the base number
    Console.Write("\nEnter a base number (0 to exit)): ");
    baseNum = Convert.ToInt32(Console.ReadLine());
}
```

Java

```java
// --------------------------------------------------------
// Write your code below this line

// Initialise variables
int baseNum = 0;
int exponent = 0;
int answer = 1;
String outString = "";

Scanner myScanner = new Scanner (System.in);

// Display the first prompt and get the base number
System.out.print("Enter a base number (0 to exit)): ");
baseNum = Integer.parseInt (myScanner.nextLine());

// Check if user wants to exit
while (baseNum != 0.0)
{
    // Display a prompt and get the exponent number
    System.out.print("Enter an exponent number: ");
    exponent = Integer.parseInt (myScanner.nextLine());

    // Check if the exponent is negative
    if (exponent < 0)
    {
        System.out.println ("Invalid exponent entered");
    }
    else
    {
        answer = 1;

        // Calculate the answer using exponentiation
        for (int i=0; i<exponent; i++)
        {
            answer = answer * baseNum;
        }

        // Create a string for the output and show the user
        outString = Integer.toString(baseNum) + " to the power of " +
                    Integer.toString(exponent) + " is " +
                    Integer.toString(answer);
        System.out.println(outString);
    }

    // Display the first prompt again and get the base number
    System.out.print("Enter a base number (0 to exit)): ");
    baseNum = Integer.parseInt(myScanner.nextLine());
}
```

Python

```
# Initialise variables
base = 0
exponent = 0
answer = 1

# Get the first base number
base = int (input ("Enter a base number (0 to exit): "))

# Check if the user wants to exit
while (base != 0):

    # Get the exponent number
    exponent = int (input ("Enter an exponent number: "))

    # Check if the exponent is negative
    if (exponent < 0):
        print ("Invalid exponent entered")
    else:
        # Reset the answer
        answer = 1

        # Calculate the answer using exponentiation
        for count in range (0, exponent):
            answer = answer * base

        # Create a string for the output sentence
        print (str (base) + " to the power of " +
               str (exponent) + " is " +
               str (answer))

    # Get another base number and lp
    base = int (input ("Enter a base number (0 to exit): "))
```

<table>
<tr><th>Question</th><th>mp</th><th>Answer</th><th>Additional Guidance</th><th>Mark</th></tr>
<tr><td>4(a)(i)</td><td>A1
A2</td><td>Award one mark for each correct cell:

<table>
<tr><th>Plaintext</th><th>Shift</th><th>Ciphertext</th></tr>
<tr><td>PIXEL</td><td>-4</td><td>LETAH</td></tr>
<tr><td>CLOUD</td><td>+3</td><td>FORXG</td></tr>
</table>
</td><td>Allow lower case ciphertext</td><td>2</td></tr>
</table>

<table>
<tr><th>Question</th><th>mp</th><th>Answer</th><th>Additional Guidance</th><th>Mark</th></tr>
<tr><td>4(a)(ii)</td><td>A3
A4</td><td>Award two marks for a comparison such as:

• Both produce same result (IQNF)(1) because the shift of -6 followed by +8 is the same as the shift of +2 (1)
• The result is the same (IQNF) (1), but the single shift is more efficient than a double shift (1)

Practical explanation:
GOLD (-6) = AIFX (+8) = IQNF (1)
GOLD (+2) = IQNF (1)</td><td>can have 1 mark from MP and 1 mark from Practical
e.g. GOLD (+2) = IQNF (1) which would be same result (1)</td><td>2</td></tr>
<tr><td>4(a)(iii)</td><td>A5
A6</td><td>Award two marks for a linked explanation such as:

The letter Y has been encrypted incorrectly as the letter V / the letter Y should have been encrypted to the letter D (1), because the +5 shift did not roll over the end of the alphabet correctly (1)</td><td><u>identify where</u> error occurred (1) and can <u>explain how</u> error occurred (1)</td><td>2</td></tr>
</table>

Question	mp	Answer	Additional guidance	Mark
4(b)	Award **one** mark for each of:		B1 does not require casting for data types on input B6 do **not** accept taking just first digit from a string B8 is only for print(), key may not be correct	**8**
	B1	Word and 2 numbers input		
	B2	Selection/loop used to check word input		
	B3	Test for word length exactly 2		
	B4	Appropriate error message for invalid string input		
	B5	Reverse the inputted word, any method		
	B6	Whole number part of the decimal number generated		
	B7	Correct key generated (concatenation)		
	B8	Key output		

Suggested test data and output			
word	**int**	**float**	**Output**
qwe/q			error message
qw	12	17.89	12wq17

C#

```
// Create the variables
String myWord = "";
String newWord = "";
int myNum = 0;
double myDecimal = 0.0;
String myKey = "";
int myWhole = 0;

// Take a word as input
Console.Write("Enter a word: ");
myWord = Console.ReadLine();

// Check that the word is correct length
if (myWord.Length == 2)
{
    // Take a whole number as input
    Console.Write("Enter a whole number: ");
    myNum= Convert.ToInt32(Console.ReadLine());

    // Take a decimal number as input
    Console.Write("Enter a decimal number: ");
    myDecimal = Convert.ToDouble(Console.ReadLine());

    // Reverse the letters in the word
    newWord = myWord[1].ToString() + myWord[0].ToString();

    // Find the whole number part of the decimal number
    myWhole = (int)myDecimal;

    // Create the new key
    myKey = String.Concat(myNum, newWord, myWhole);

    // Display the new key for the user
    Console.Write("The key is " + myKey);
}
else
{
    // Word is not the correct length, so display an error message
    Console.WriteLine("Invalid word entered");
}
```

Java

```
    // Write your code below this line

    Scanner myScanner = new Scanner (System.in);

    // Create the variables
    String myWord = "";
    String newWord = "";
    int myNum = 0;
    Double myDecimal = 0.0;
    String myKey = "";
    int myWhole = 0;

    // Take a word as input
    System.out.print("Enter a word: ");
    myWord = myScanner.nextLine();

    // Check that the word is correct length
    if (myWord.length() == 2)
    {
        // Take a whole number as input
        System.out.print("Enter a whole number: ");
        myNum = Integer.parseInt(myScanner.nextLine());

        // Take a decimal number as input
        System.out.print("Enter a decimal number: ");
        myDecimal = Double.parseDouble (myScanner.nextLine());

        // Reverse the letters in the word
        newWord = Character.toString(myWord.charAt(1)) +
                  Character.toString(myWord.charAt(0));

        // Find the whole number part of the decimal number
        myWhole = myDecimal.intValue();

        // Create the new key
        myKey = myNum + newWord + myWhole;

        // Display the new key for the user
        System.out.println("The key is " + myKey);
    }
    else
    {
        // Word is not the correct length, so display an error message
        System.out.println("Invalid word entered");
    }
}
```

Python

```
# Take a two-letter word as input
myWord = input ("Enter a word: ")

# Check that the word is correct length
if (len (myWord) == 2):

    # Take a whole number as input
    myNum = int (input ("Enter a whole number: "))

    # Take a decimal number as input
    myDecimal = float (input ("Enter a decimal number: "))

    # Reverse the letters in the word
    newWord = myWord[1] + myWord[0]

    # Find the whole number part of the decimal number
    myWhole = int (myDecimal)

    # Create the new key
    myKey = str (myNum) + newWord + str (myWhole)

    # Display the new key for the user
    print (myKey)

# Word is not the correct length, so display an error message
else:
    print ("Invalid word entered")
```

<table>
<tr><th>Question</th><th>mp</th><th>Answer</th><th>Additional Guidance</th><th>Mark</th></tr>
<tr>
<td>5(a)</td>
<td>A1
A2</td>
<td>Award 1 mark for demonstration of one correctly merged/split row
Award 2 marks for correct merge sort

<table>
<tr><td>Blackfin, Bigeye</td><td>Longtail, Albacore</td><td>Bluefin</td></tr>
</table>

<table>
<tr><td>Longtail, Blackfin, Bigeye, Albacore</td><td>Bluefin</td></tr>
</table>

<table>
<tr><td>Longtail, Bluefin, Blackfin, Bigeye, Albacore</td></tr>
</table>

OR

<table>
<tr><td>Bigeye</td><td>Blackfin, Albacore</td><td>Longtail, Bluefin</td></tr>
</table>

<table>
<tr><td>Bigeye</td><td>Longtail, Bluefin, Blackfin, Albacore</td></tr>
</table>

<table>
<tr><td>Longtail, Bluefin, Blackfin, Bigeye, Albacore</td></tr>
</table>

OR

<table>
<tr><td>Blackfin, Bigeye</td><td>Albacore</td><td>Longtail, Bluefin</td></tr>
</table>

<table>
<tr><td>Blackfin, Bigeye</td><td>Longtail, Bluefin, Albacore</td></tr>
</table>
</td>
<td>We are testing pupils understanding of how a merge sort progresses:

Divide and conquer (split)
Merging adjacent elements

Allow answers in ascending order</td>
<td>2</td>
</tr>
</table>

		Longtail, Bluefin, Blackfin, Bigeye, Albacore OR Blackfin, Bigeye | Albacore | Longtail, Bluefin Blackfin, Bigeye, Albacore | Longtail, Bluefin Longtail, Bluefin, Blackfin, Bigeye, Albacore		

<table>
<tr><th>Question</th><th>mp</th><th>Answer</th><th>Additional Guidance</th><th>Mark</th></tr>
<tr><td>5(b)</td><td>B1
B2</td><td>Award one mark for each correct cell:
<table><tr><td>Line number with error</td><td>11 (1)</td></tr><tr><td>Corrected line of pseudocode</td><td>SET tmp TO myTuna[ndx] (1)</td></tr></table></td><td>Ignore spellings

Ignore missing text as long as the minimum of [ndx] is provided

B2 does not depend on B1</td><td>2</td></tr>
</table>

Question	mp	Answer	Additional guidance	Mark
5(c)	Award **one** mark for each of:		C1 allow append as writing to file C4 Do not award comma within an array C5 must have 1 complete line written to file (allow additional spaces and commas). Should include individual elements (2D index) and not a 1D array from tbl_tuna C6 complete file must be as expected e.g. no additional comma at end of line and no spaces around commas	**6**

C#

```
// Q05cFINISHED

using System.IO;

String[,] tblTuna =  {
    { "Yellowfin", "105", "15", "3"},
    { "Albacore","90","15","5"},
    { "Skipjack","50","3","4"},
    { "Bigeye","105","25","4"},
    { "Atlantic Bonito","50","4","2"},
    { "Northern Bluefin","190","120","11"},
    { "Southern Bluefin","190","120","11"},
    { "Tongol","90","20","4"}
};



// ---------------------------------------------
// Write your code below this line

String fileName = "TunaData.txt";       // Output file name
String comma = ",";                     // Use as a constant
String full_path = "C:\\Q05c";
fileName = full_path + "\\" + fileName;

// Create variables as needed
int number = 101;           // Number for the line at the front
String lineOut = "";        // The line to output

// Get the dimensions of the array for looping
int rows = tblTuna.GetLength(0);
int columns = tblTuna.GetLength(1);

// Open the file for writing
StreamWriter fileWriter = new StreamWriter(fileName);

```

```
// Process every tuna in the table
for (int i = 0; i < rows; i++)          // Rows
{
    // Start with the number and a comma
    lineOut = number.ToString() + comma;

    for (int j = 0; j < columns; j++)   // Columns
    {
        // Add the name and the numbers
        lineOut = lineOut + tblTuna[i, j];

        // Add a comma to all the fields except the last
        if (j < columns - 1)
        {
            lineOut = lineOut + comma;
        }
    }
    // Write the line to the file
    fileWriter.WriteLine(lineOut);

    // Go to the next number for the line
    number = number + 1;
}

// Close the file
fileWriter.Close();
```

Java

```java
// Q05cFINISHED

import java.io.FileWriter;
import java.io.IOException;

public class Q05cFINISHED {

    public static void main(String[] args) throws Exception {

        String[][] tblTuna =
        {
            { "Yellowfin", "105", "15", "3"},
            { "Albacore","90","15","5"},
            { "Skipjack","50","3","4"},
            { "Bigeye","105","25","4"},
            { "Atlantic Bonito","50","4","2"},
            { "Northern Bluefin","190","120","11"},
            { "Southern Bluefin","190","120","11"},
            { "Tongol","90","20","4"}
        };

        // ------------------------------------------
        // Write your code below this
        String fileName = "TunaData.txt";       // Output file name
        String comma = ",";                     // Use as a constant
        String full_path = "C:\\src\\q05c";
        fileName = full_path + "\\" + fileName;

        // Create variables as needed
        int number = 101;           // Number for the line at the front
        String lineOut = "";        // The line to output
        String linefeed = "\n";     // Use as a constant

        // Get the dimensions of the array for looping
        int rows = tblTuna.length;
        int columns = tblTuna[0].length;

        // Open the file for writing
        FileWriter fileWriter = new FileWriter(fileName);

```

```
41          // Process every tuna in the table
42          for (int i = 0; i < rows; i++)          // Rows
43          {
44              // Start with the number and a comma
45              lineOut = Integer.toString(number) + comma;
46
47              for (int j = 0; j < columns; j++)   // Columns
48              {
49                  // Add the name and the numbers
50                  lineOut = lineOut + tblTuna[i][j];
51
52                  // Add a comma to all the fields except the last
53                  if (j < columns - 1)
54                  {
55                      lineOut = lineOut + comma;
56                  }
57                  else
58                  {
59                      lineOut = lineOut + linefeed;
60                  }
61              }
62              // Write the line to the file
63              fileWriter.write(lineOut);
64
65              // Go to the next number for the line
66              number = number + 1;
67          }
68
69          // Close the file
70          fileWriter.close();
71      }
72
73  }
```

Python

```python
# Q05cFINISHED

tbl_tuna = [["Yellowfin",105,15,3],
            ["Albacore",90,15,5],
            ["Skipjack",50,3,4],
            ["Bigeye",105,25,4],
            ["Atlantic Bonito",50,4,2],
            ["Northern Bluefin",190,120,11],
            ["Southern Bluefin",190,120,11],
            ["Tongol",90,20,4]]

# ------------------------------------------------
# Write your code below this line

FILE_OUT = "TunaData.txt"           # Output file name
COMMA = ","                         # Use as constant

# Create variables as needed
LINEFEED = "\n"
number = 101                        # Number at the front
line_out = ""                       # The line to write

# Open the file for writing
file = open (FILE_OUT, "w")

# Process every tuna in the table
for tuna in tbl_tuna:
    # Start with the number and a comma
    line_out = str (number) + COMMA

    # Add the name of the tuna and a comma
    line_out = line_out + tuna[0] + COMMA

    # Each of the numbers need to be converted to a string
    # before adding to the output string.
    line_out = line_out + str (tuna[1]) + COMMA
    line_out = line_out + str (tuna[2]) + COMMA
    line_out = line_out + str (tuna[3])     # No comma on last field

    # Add a line feed to the whole line
    line_out = line_out + LINEFEED

    # Write the line to the file
    file.write (line_out)

    # Go to the next number for the line
    number = number + 1

# Close the file
file.close ()
```

Question	mp	Answer	Additional guidance	Mark
6	Award **one** mark for each of:			
	A1	Any variable for tracking best breed	Initial values could be the first breed in the table	
	A2	A loop to all items in an array	The same index can be used across all the individual arrays, regardless of loop type	
	A3	Daily volume calculation is volume * count	Disregard accuracy of indexing	
	A4	Rows of arrays are displayed	Disregard presence/lack of calculated daily volume	
	A5	Total volume calculated correctly by any method		
	A6	Relational operators used to compare rating and daily volume		
	A7	Boolean operator/nested selection used to combine tests		
	A8	Add calculated daily volume to the new data structure		
	A9	Description of data fields displayed (no alignment required)		
	A10	Recommended breed identified correctly	by any method except hard-coded	
	A11	Outputted information is fit for purpose and suitable for the audience		**11**

Award up to a maximum of nine marks using the levels-based mark scheme below.				Mark
Band 0	**Band 1 (1-3 marks)**	**Band 2 (4-6 marks)**	**Band 3 (7-9 marks)**	
No rewardable content	Little attempt to decompose the problem into component parts	Some attempt to decompose the problem into component parts	The problem has been decomposed into component parts	
	Some parts of the logic are clear and appropriate to the problem	Most parts of the logic are clear and mostly appropriate to the problem	The logic is clear and appropriate to the problem	
	Some appropriate use and manipulation of data types, variables, data structures and program constructs	The use and manipulation of data types, variables and data structures and program constructs is mostly appropriate	The use and manipulation of data types, variables and data structures and program constructs is appropriate	
	Parts of the code are clear and readable	Code is mostly clear and readable	Code is clear and readable	
	Finished program will not be flexible enough with other data sets or input	Finished program will function with some but not all other data sets or input	Finished program could be used with other data sets or input	
	The program meets some of the given requirements	The program meets most of the given requirements	The program fully meets the given requirements	**(9)**

C#

```
// Q06FINISHED

string[] tbl_breed = { "Red Chittagong", "Sussex", "Dexter",
                       "Abondance", "Sahiwal", "Vorderwald",
                       "Ayrshire", "Jersey", "Randall",
                       "Alderney", "Carora", "Gloucester"};
int[] tbl_rating = { 1, 2, 3, 2, 3, 1, 2, 1, 2, 1, 3, 2 };
int[] tbl_count = { 6, 3, 8, 7, 6, 4, 3, 7, 3, 3, 4, 7 };
double[] tbl_volume = {7.5, 5.7, 11.4, 11.4,
                       22.0, 15.2, 21.0, 18.3,
                       19.0, 9.0, 23.1, 16.0 };
double[] tbl_dailyVolume = {0.0, 0.0, 0,0, 0,0,
                            0.0, 0.0, 0.0, 0.0,
                            0.0, 0.0, 0.0, 0.0};

// ------------------------------------------------
// Write your code below this line

const String SPACE = " ";           // Just for reading

// Variables for indexing and totals
double totalVolume = 0.0;
double todayVolume = 0.0;

// Variables for outputting
string outString = "";

// Initialise the maximum values to the first instance
int maxIndex = 0;

// Display a message to describe the data fields
Console.WriteLine ("Fields are: Breed, Rating, Volume (cow), Count, Volume (day)");
```

```
// Process every breed of cow
for (int i = 0; i < tbl_breed.Length; i++)
{
    // Calculate volume per day
    todayVolume = tbl_volume[i] * tbl_count[i];

    // Add today's volume to the daily volume table
    tbl_dailyVolume[i] = todayVolume;

    // Display the row of information for this breed
    outString = tbl_breed[i] + SPACE +
                tbl_rating[i].ToString() + SPACE +
                tbl_volume[i].ToString() + SPACE +
                tbl_count[i].ToString() + SPACE +
                tbl_dailyVolume[i].ToString();
    Console.WriteLine(outString);

    // Keep a running total of milk production
    totalVolume = totalVolume + todayVolume;

    // Test for a best breed
    if ((tbl_rating[i] <= tbl_rating[maxIndex]) &
        (tbl_volume[i] >= tbl_volume[maxIndex]))
    {
        maxIndex = i;
    }
}

// Display the total volume of milk in a day
outString = "Total: " + totalVolume.ToString() + " litres";
Console.WriteLine(outString);

// Display the recommended breed
outString = "Recommended breed: " + tbl_breed[maxIndex] +
            " rating: " + tbl_rating[maxIndex].ToString() +
            " volume: " + tbl_volume[maxIndex];
Console.WriteLine(outString);
```

Java

```java
// Q06FINISHED

public class Q06FINISHED
{
    static final String SPACE = " ";            // Just for reading

    public static void main(String[] args)
    {
        String[] tbl_breed = { "Red Chittagong", "Sussex", "Dexter", "Abondance",
                               "Sahiwal", "Vorderwald", "Ayrshire", "Jersey",
                               "Randall", "Alderney", "Carora", "Gloucester"};
        int[] tbl_rating = { 1, 2, 3, 2, 3, 1, 2, 1, 2, 1, 3, 2 };
        int[] tbl_count = { 6, 3, 8, 7, 6, 4, 3, 7, 3, 3, 4, 7 };
        double[] tbl_volume = { 7.5, 5.7, 11.4, 11.4,
                                22.0, 15.2, 21.0, 18.3,
                                19.0, 9.0, 23.1, 16.0 };
        double [] tbl_dailyVolume = {0.0, 0.0, 0,0, 0,0,
                                     0.0, 0.0, 0.0, 0.0,
                                     0.0, 0.0, 0.0, 0.0};

        // ---------------------------------------------
        // Write your code below this line

        // Variables for indexing and totals
        double totalVolume = 0.0;
        double todayVolume = 0.0;

        // Variables for outputting
        String outString = "";

        // Initialise the maximum values to the first instance
        int maxIndex = 0;

       // Display the titles in a table
        System.out.println ("Fields are: Breed, Rating, Volume (cow), Count, Volume (day)");

```

```
        // Process every breed of cow
        for (int i = 0; i < tbl_breed.length; i++)
        {
            // Calculate volume per day
            todayVolume = tbl_volume[i] * tbl_count[i];

            // Add today's volume to the daily volume table
            tbl_dailyVolume[i] = todayVolume;

            // Display the row of information for this breed
            outString = tbl_breed[i] + SPACE +
                        Integer.toString(tbl_rating[i]) + SPACE +
                        String.valueOf(tbl_volume[i]) + SPACE +
                        Integer.toString(tbl_count[i]) + SPACE +
                        String.valueOf(tbl_dailyVolume[i]);
            System.out.println (outString);

            // Keep a running total of milk production
            totalVolume = totalVolume + todayVolume;

            // Test for a best breed
            if ((tbl_rating[i] <= tbl_rating[maxIndex]) &
                    (tbl_volume[i] >= tbl_volume[maxIndex]))
            {
                maxIndex = i;
            }
        }

        // Display the total volume of milk in a day
        outString = "Total: " + Double.toString(totalVolume) + " litres";
        System.out.println (outString);

        // Display the recommended breed
        outString = "Recommended breed: " + tbl_breed[maxIndex] +
                    " rating: " +
                    Integer.toString(tbl_rating[maxIndex]) +
                    " volume: " +
                    Double.toString(tbl_volume[maxIndex]);
        System.out.println (outString);
    }
}
```

Python

```
# Q06FINISHED

tbl_breed = ["Red Chittagong", "Sussex", "Dexter", "Abondance",
             "Sahiwal", "Vorderwald", "Ayrshire", "Jersey",
             "Randall", "Alderney", "Carora", "Gloucester"]
tbl_rating = [1, 2, 3, 2, 3, 1, 2, 1, 2, 1, 3, 2]
tbl_count = [6, 3, 8, 7, 6, 4, 3, 7, 3, 3, 4, 7]
tbl_volume = [7.5, 5.7, 11.4, 11.4,
              22.0, 15.2, 21.0, 18.3,
              19.0, 9.0, 23.1, 16.0]
tbl_dailyVolume = []

# ---------------------------------------------
# Write your code below this line

# Variables for indexing and totals
index = 0
totalVolume = 0.0

# Initialise the maximum values to the first instance
maxIndex = 0

# Display a message to describe the data fields
print ("Fields are: Breed, Rating, Volume (cow), Count, Volume (day)")

# Process every breed of cow
for index in range (len (tbl_breed)):
    # Calculate volume per day
    todayVolume =  tbl_volume[index] * tbl_count[index]

    # Add today's volume to the daily volume table
    tbl_dailyVolume.append (todayVolume)

    # Display the row of information for this breed
    print (tbl_breed[index],tbl_rating[index],
           tbl_volume[index], tbl_count[index], tbl_dailyVolume[index])

    # Keep a running total of milk production
    totalVolume = totalVolume + todayVolume

    # Test for a best breed
    if ((tbl_rating[index] <= tbl_rating[maxIndex]) and
                (tbl_volume[index] >= tbl_volume[maxIndex])):
        maxIndex = index

# Display the total volume of milk in a day
layout = "Total: {} litres"
print (layout.format (totalVolume))

# Display the recommended breed
print ("Recommended breed: " + tbl_breed[maxIndex],
       "rating: " + str(tbl_rating[maxIndex]),
       "volume " + str(tbl_volume[maxIndex]))
```

Pearson Education Limited. Registered company number 872828
with its registered office at 80 Strand, London, WC2R 0RL, United Kingdom